PATENT APPLICATION
DOCKET NO.: PXL-045

PRINTER DRIVER EXTENSION AND RELATED METHODS

Field of the Invention

[0001] The invention relates generally to printer drivers and more specifically to an extension for an industry standard printer driver.

Background of the Invention

[0002] Generally, there are two types of printer drivers -- monolithic drivers and table-driven drivers. Monolithic drivers are custom pieces of software code that translate the print job from computer application language to a format readable by a printer. Monolithic drivers require a large amount of software code and thus are costly to develop. Alternatively, table-driven drivers were developed because each driver typically performs a similar set of functions. Although the commands for performing certain functions may be different for various, there is a common set of operations that are accomplished for each print job.

[0003] One example of a table-driven driver is the Universal Printer Driver (UNIDRV) provided as part of the WINDOWS™ operating system. A very basic UNIDRV solution can be implemented without writing any code. To control the behavior of the printer, a printer driver solution provides a user-interface (UI) having dialog boxes and property sheets. The UI allows a user to control how output appears on the printed page and provides the user with printing options, such as paper sizes that are supported by the printer. Typically, printer manufacturers control the look and feel of the UI. Although the UI supplied with UNIDRV, known as the "treeview," can be customized to some extent, the overall look and feel of the UI can not be readily manipulated. As shown in FIG. 1A, the treeview is a structured list of settings that aligns

the settings horizontally at predetermined positions. Another limitation of the UNIDRV is that is does not filter the options that are shown in the treeview. For example, all paper formats and sizes are shown in the treeview, even if the connected printer does not actually supports those formats and sizes. For example, the option to print on A4 paper is provided to the user even if the printer cannot print on A4 paper.

[0004] Another limitation of the UNIDRV is the lack of the ability to provide color management dependant upon whether text or graphics is being printed. If the line being printed contains both text and graphics, UNIDRV prints all black pixels using a combination of CMYK instead of using pure black to print the text portions of the line. Similarly, UNIDRV only allows a single half-tone screen for each line of printing. If the line contains both text and graphics, a smooth half-tone screen is applied to all pixels, even though it is desirable to apply a detail half-tone screen to the text and vector graphics portions of the print job

[0005] Additionally, most commercial printers are shipped with a resident hypertext transport protocol (HTTP) server. This server provides printer status information and requires a complex USBDOT4 protocol used in conjunction with a custom I/O driver resident on both the client computer and the printer. Serving HTML pages through this interface increases the amount of traffic through the interface, which, in turn, slows responsiveness.

[0006] Therefore, a need exists for an extension of the UNIDRV that allows for the customization of the UI, facilitates the ability to control the options presented in the UI, that provides various printing enhancement features such as multiple forms of color management and the use of multiple half-tone screens on a pixel-by-pixel basis, and that provides printer status information.

Summary of the Invention

[0007] The present invention provides an extension to the UNIDRV solution that allows a manufacturer to customize the look and feel of the UI. The extension improves the speed of printer status reports and improves the overall printer performance by facilitating the use of color management and the application of multiple half-tone screens on a pixel-by-pixel basis.

[0008] In one aspect, the invention is directed to a method of customizing a standard user interface associated with a universal printer drive. The method includes the steps of associating an item of a standard user interface data structure to a first object, associating an item of a customized user interface to a second object, linking the first object to the second object through a software interface, and displaying the customized user interface. The software interface facilitates communication between the first object and the second object,.

[0009] In various embodiments, the step of displaying includes accessing a definition file. The definition file includes information related to the customized user interface. The information related to the customized user interface can include at least one additional item compatible with the standard user interface structure.

[0010] The method can also include the step of filtering at least one item of the standard user interface data structure prior to the displaying step. The filtering step includes writing to a file. The file includes data related to a state of at least one constant. The state of the at least one constant being determinative of inclusion in the standard user interface data structure.

[0011] In another aspect, the invention is directed to computer software, residing on a computer-readable storage medium, including a set of instructions that cause a computer to customize a standard user interface associated with a universal printer driver by associating an item of a standard user interface data structure to a first object, associating an item of a

customized user interface to a second object, linking the first object to the second object through a software interface, and displaying the customized user interface. The software interface facilitates communication between the first object and the second object.

[0012] In another aspect, the invention is directed to a method of extending rendering functionality of a standard universal printer driver. The method includes the steps of generating a tagging bitmap and intercepting a drawing call to a banding bitmap of the standard universal driver. The tagging bitmap has substantially similar boundaries as a banding bitmap. The banding bitmap is used in rendering image information. The drawing call includes a drawing function and an object type related to the drawing function. The method also includes the steps of storing the object type associated with the drawing call in the tagging bitmap, performing error correction of the object type stored in the tagging bitmap, and incorporating the object type stored in the tagging bitmap with the image information of the banding bitmap to render a final output.

[0013] In various embodiments, the method includes the step of preprocessing the image information of the banding bitmap by alpha-blending a watermark image with the image information. The step of performing error correction includes performing error correction for raster operation functions.

[0014] In another embodiment, the step of storing includes storing information related to a half-tone filter. The information related to the half-tone filter includes information determinative of the half-tone filter to apply to the image information on a pixel-by-pixel basis. The step of storing can also include storing information related to color management, such as converting from an input color space to an output color space on a pixel-by-pixel basis and black-

generation. In additional embodiments, the information stored in the tagging bitmap facilitates white space skipping and transition determination.

[0015] In another aspect, the invention is directed to a method of providing printer status information. The method includes the steps of using a field in a printer status page to generate a simple network management protocol request; communicating the simple network management protocol request to a simple network management protocol server resident on at least one of a remote printer and a remote printer server; translating the simple network management protocol response provided by the at least one of the remote printer and the remote printer server into a format usable by the printer status page, and inserting the translated simple network management protocol response into the field of the printer status page.

[0016] In one embodiment, the printer status page is viewable by a web browser and is comprised of hypertext markup language.

[0017] In another embodiment, communicating step includes communicating the simple network management protocol request from a computer to the at least one of the remote printer and the remote printer server using a standard universal serial bus driver.

[0018] In an alternative embodiment, the communicating step includes communicating the simple network management protocol request from a computer to the at least one of the remote printer and the remote printer server using a standard networking protocol. The standard networking protocol can be selected from the group consisting of internetwork packet exchange, transition control protocol/internet protocol, file transfer protocol, and user datagram protocol.

Brief Description of the Drawings

[0019] The invention is pointed out with particularity in the appended claims. The advantages of the invention may be better understood by referring to the following description taken in conjunction with the accompanying drawing in which:

[0020] FIG. 1 is a prior art diagram of the treeview associated with the UNIDRV;

FIG. 1A is a graphical representation of the treeview;

FIG. 1B is a graphical representation of an embodiment of a customized user interface constructed in accordance with the principles of the invention ;

FIG. 2 is a flow chart of depicting the steps of creating a customized user interface;

FIG. 3 is a flow chart of depicting the steps of extending the rendering functionality; and

FIG. 4 is block diagram of an embodiment of a printing device status system according tot he principles of the present invention.

Detailed Description of the Invention

[0021] The present invention extends the functionality of a standard WINDOWS printer driver, (e.g., the UNIDRV or the PSCRIPT) by providing the ability to customize the UI, enhance the printer performance by providing the ability to use multiple half-tone screens and multiple forms of color management, and to display printer status without the need for an HTTP server resident at the printer. In one embodiment, the present invention is a software "plug-in" module configured to provide at least one of these features. The plug-in communicates with the core UNIDRV code provided as part of the operating system. The IPrintOemUni COM interface facilitates communication between the core UNIDRV code and the software plug-in of the present invention.

[0022] FIG. 1A depicts one embodiment of a treeview 10 of the UNIDRV. The standard UI 10 contains a list of control parameters (20A, 20B, 20C, referred to generally as 20) that are aligned horizontally at predetermined stop-tab positions. Examples of control parameters 20 can include, but are not limited to, paper/output, graphic, and document options. Each of the control parameters 20A, 20B, 20C, can have additional sub-parameters (20AA, 20AB, 20BA, 20BB, 20CA, ...). Examples of sub-parameters can include, but are not limited to, paper size, copy count, print quality, true type font, advanced printing features, half toning, print optimization, and printer features. The sub-parameters can also have their own sub-parameters (20CB1, 20CB2, ...) such as, economode and graphics mode. Each of the parameters controls a different aspect of the printer driver functionality.

[0023] To change the settings of one of the parameters 20, a user selects the specific parameter with a mouse or similar device. The appearance of the parameter changes from a display–only style to a data selection style. For example, the parameter 20 can become a combo-box 30. The combo-box 30 contains a list of entries that are selectable for the parameter. For example, if paper size is selected the combo-box can contain a list that includes A4, legal, and letter.

[0024] FIG. 1B depicts an embodiment of a customized UI 50. The customized UI includes a dialog box 52 and a plurality of dialog page tabs 54A, 54B, 54C, 54D, (referred to generally as 54). Selecting a respective page tab 54 displays configurable options to a user of the customized UI. For example, selecting the "basic" tab 54A displays configurable items 60A, 60B (referred to generally as items 60) to the user. The items 60 facilitate control of certain aspects of the printer, e.g., number of copies and the orientation of the printed page. The item 60 can be manipulated by radio buttons, combo-boxes, and other well know methods.

[0025] With reference to FIGS. 1A, 1B, and 2, the standard UI 10 provided as part of the UNIDRV is replaced with a customizable UI 50. In more detail, each parameter 20 of the treeview 10 is a piece of software code known as an OPTITEM. An OPTITEM is a C programming language structure that is used by common property sheet user interface (CPSUI) applications (including printer interface DLLs) for describing one property sheet option on a property sheet page. In one embodiment of the present invention, each OPTITEM is inspected (STEP 100). There are various forms of OPTITEMs. Two typical examples are lists, which contain one or more entries, and values, which are typically alphanumeric in nature. Based on the category of the OPTITEM, and object of a class OPTITEMHANDLER is created (STEP 110). Each OPTITEMHANDLER instance is associated with a respective OPTITEM of the treeview 10 and can communicate with the OPTITEM to perform certain functions, e.g., get/set values, enumerate all entries, enable, disable, hide and show. Also, each instance of the OPTITEMHANLDER class includes a standard interface configured to allow a client of the OPTITEMHANDLER access to the OPTITEMHANDLER in a unified manner. As such, a layer is created that allows management of the OPTITEMs in the treeview 10. A variety of actions can be performed on the OPTITEMs of the treeview 10 as a result of this layer (e.g., get/set a current value, enumerate all items in a list, enable/disable a single item in a list, enable/disable an item, remove an entry of a list, and hide/show OPTITEMs) without actually changing the OPTITEM by selecting the OPTITEM in the treeview 10.

[0026] Each item 60 of the customized UI 50 can also be represented as a piece of software code. Based on the category of the item 60 an object of a class DlgLink is created (STEP 120). Each object is associated with a respective item of the customized UI 50 and can communicate with the item 60 to perform certain functions, e.g., get/set values, enumerate all entries, enable,

disable, hide and show. Also, each instance of the DlgLink class includes a standard interface configured to allow a client of the object to access to the object in a unified manner. In this manner, a layer is created that allows management of the items in the customized UI 50.

[0027] Each of the respective OPTITEMHANLDER objects is linked to an object associated with an item 60 of the customized UI (STEP 130). In other words, each parameter 20 of the treeview 10 is linked to an item 60 of the customized UI 50 via a software interface. The link can be established by a fixed assignment of an OPTITEM ID to a customized UI control ID. Alternatively, the link can be established dynamically by loading linking information from a file, such as, an object definition file (ODF), which is described in more detail below. The software interface isolates the implementation details of the treeview 10 and the implementation details of the customized UI 50 from each other. This separation enables constraint logic and other user interface logic to operate on the hierarchy of treeview parameters 20 without creating dependencies related to how the controls are represented in the customized UI 50.

[0028] Once each OPTITEM of the treeview 10 is linked to an appropriate customized UI control 60, the standard UI can be replaced. In addition to the treeview 10, the UNIDRV generates two standard dialog pages "Layout" and "Paper/Quality". These pages and the treeview 10 are hidden from the user when the customized user interface is created (STEP 140). These pages are still generated by the UNIDRV; however, they are not displayed to the user. Instead, only the customized user interface is shown. More specifically, the state of all of the OPTITEMs in the treeview 10 is set to "hidden", except for the "Orientation" control, which is used to specify portrait or landscape mode. In response, the UNIDRV automatically hides the "Paper/Quality" tab. In contrast, the "Layout" tab generated by the UNIDRV can not be hidden. As a result, the existing dialog box generated by the UNIDRV must be redefined. More

specifically, the Title/Caption text of the dialog box is replaced different text name. Also, all UI controls contained in the layout tab are redefined to an off-screen location so they can not be seen by the user. Then, the desired customized controls are created dynamically and arranged on the now empty layout tab. The layout tab control needs to be present in order to not disturb the internal operations of the UNIDRV. The additional tabs can be created and displayed in a standard manner.

[0029] In addition to replacing the standard UI with a custom UI, in one embodiment the present invention facilitates control of the extended functionality of the UNIDRV described herein through the use of the ODF. The ODF provides the ability to define additional UI elements and their respective settings. The ODF is a text based parameter file. Each driver created is associated with a specific ODF. For example, if a computer is able to print to both a laser printer and an ink jet printer a different driver is required to print to the respective printers. The present invention facilities the creation and control of a different UI for each respective driver. As such, an ODF for each driver is required. While described as a stand-alone file, the ODF can be incorporated into a file included with the UNIDRV as described in more detail below.

[0030] The parameters defined in the ODF are compiled at driver installation. The complied form of the ODF is stored with the other printer driver settings in the Windows registry. This facilities access to the information contained in the ODF. Examples of entries in the ODF can include data structures, additional OPTITEMs, UI behavior of the additional OPTITEMs, and the UI behavior of the driver.

[0031] In one embodiment, the invention includes the ability to filter the printer features reported to an application, e.g., Word, Excel, or PowerPoint. The UNIDRV includes a generic

printer description (GPD), which contains information about the printer features that UNIDRV presents in the UI to allow the user to configure the printer. The GPD file can also include the information contained in an ODF as described above. For example, a GPD file may specify five different paper formats: A4, LETTER, LEGAL, B4, A3. The GPD file may also specify two different paper feeder options: option 1 contains a MANUAL source and option 2 contains a tray-1 source. For a given printer, the manual feeder can accept all papers sizes and tray-1 can only hold A4, LETTER, and LEGAL paper sizes. The UNIDRV does not display a filtered set of choices to the user if the user selects tray-1 as the paper source. Instead, the user will be allowed to select a paper size of B4 even though tray-1 does not support that option.

[0032] In one embodiment of the invention, a GPD "include" file is created that defines which options are displayed to the user. The include file contains constants that are determinative of whether or not an option is displayed in the customized UI 60. The use of the include file provides advantages such as dynamic form to tray assignment in the customized UI, dynamic media type to tray assignment in the customized UI, and the ability to enable and disable document preference features in response to the printer property features.

[0033] Another aspect of the invention extends the rendering functionality of the UNIDRV. Rendering refers to actions and procedures that are necessary to control the creation of print jobs, printed pages, and the actual imaging of objects on the created pages. Additionally, rendering relates to color management, half toning, and compression. Object specific half toning and color management require that tagging information be associated with each pixel of a final printed product. As mentioned above, the UNIDRV does not use object tagging information during the rendering process.

[0034] With reference to FIG. 3, in one embodiment of the present invention an object type tagging bitmap is created (STEP 300). The object type tagging bitmap is substantially similar in size as a banding bitmap used by the UNIDRV to render each band of the imagining information for the final printed page. Additionally, various brushes are created. Brushes are descriptions of objects that are used to fill areas of the object type tagging bitmap. Traditionally, brushes hold a specific color value. In the present invention, the brushes contain bitwise information or a combination of all the different type tag bits used. For example, the object type tagging information can be related to a specific form of black generation or a specific form of half-toning. The tagging information can also be used in edge transition determination processes and also for white-space skipping. In one embodiment, an additional four bits of information is stored in the object type tagging bitmap for each pixel of the final product; however, additional bit sizes can be used.

[0035] Drawing calls that are made to the banding surface are intercepted (STEP 310). The actual drawing operations are not modified or changed. Each drawing call includes a drawing function and an object type related to that drawing function. The object type information associated with the drawing function is stored in the object type tagging bitmap (STEP 320). In one embodiment, the same input parameters of the DRV drawing call are passed to the analogous ENG function, which records the object type information in the object type tagging bitmap.

[0036] For example, a routine to intercept a drawing call can be implemented according to the following pseudo-code:

```
DrvBitBlt( SOURCE, PATTERN, DESTINATION, COORDINATES, ROP )
{
        Is SOURCE present and ROP includes SOURCE ?
```

{

Mark object tag bitmap according to error correction rules described in more detail below with image-specific marking brush (use COORDINATES to specify area). Use EngBitBlt to mark.

}

else

{ // Deprecated special case

Mark object tag bitmap according to rules set forth in d) with vector-specific marking brush (use COORDINATES to specify area). Use EngBitBlt to mark.

}

UNIDrvBitBlt(SOURCE, PATTERN, DESTINATION, COORDINATES, ROP)

}

[0037] In one embodiment, the plug-in provides a set of user defined drawings functions that replace the equivalent drawing function provided by the UNIDRV. The set of drawing functions can include, for example, DrvAlphaBlend, DrvBitBlt, DrvCopyBits, DrvFillPath, DrvGradientFill, DrvLineTo, DrvPaint, DrvPlgBlt, DrvStretchBlt, DrvStretchBltROP, DrvStrokeAndFillPath, DrvStrokePath, DrvTextOut, and DrvTransparentBlt. Pseudo-code for one of these function can be implemented as follows:

// XXXX: is placeholder for any of the above function names, e.g. BitBlt

// YYYY: placeholder for any one of brushProcessSmooth, brushProcessLineArt, etc..., selected based upon the type of function XXXX that is implemented

DrvXXXX(parameters)

{

```
        // first store the object type tag bits
        DevOEM->ObjectTypeTagXXXX( parameters, brushYYYY );
        // Now call original UNIDRV entry point to do actual object  drawing
        fpUNIDRV_XXXX( parameters );
        }
DevOEM::ObjectTypeTagXXXX( parameters, brush )
        {
        // perform error correction as described in more detail below
        correctedROPorMIX= DevOEM->FixupROP( parameters );
        // Now call EngXXXX Drawing API to record object type info to the tagging bitmap
        EngXXXX( hObjectTag, parameters, correctedROPorMIX, brush);
        }
```

[0038] Error correction is performed (STEP 330) to ensure that the object type information stored in the tagging bitmap yields a meaningful result when incorporated with the image information to generate the final output. In one embodiment, the error correction is related to raster operations (ROP) and/or MIX mode requests. A ROP or MIX mode is a set of instructions that perform binary logical operations while drawing to the target surface. The logical operation performed by the ROP receives input parameters such as D (an existing pixel on the target surface), S (a pixel on the source surface), and P (a pixel in an arbitrary fill pattern), which are well known to those of ordinary skill in the art. Raster operations are often performed in sequences. Some of these operations can modify bits on the target surface in an unexpected fashion. As result, error correction is needed to ensure the object type information stored in the tagging bitmap generates a useful result.

[0039] To create the final output, the UNIDRV core code calls the IPrintOemUni::ImageProcessing function, which integrates the image information from the banding bitmap with the object type information stored in the tagging bitmap (STEP 340). In addition, a watermark is combined with the image information through alpha-blending. For example, the function can be implemented according to the following pseudo-code:

For (each pixel (x,y))

[0040]
```
{
    char c, m, y, k;      // Resulting pixels, on(1) or off(0)
    long cmykValue;  // Variable that holds color conversion result
    long rgbValue= GetPixel(x,y) * GetWatermarkPixel (x,y)
    int typetag= DevOEM->GetObjectTypeTag(x,y)
    // Do type specific color conversion by calling appropriate function
    if ( typetag & OBJMASK_4PLANE_BLACK )
        cmykValue= RGB_To_CMYK_ProcessBlack( rgbValue )
    else
        cmykValue= RGB_To_CMYK_BlackOnly( rgbValue );
    if ( typetag & OBJMASK_SMOOTH_HALFTONE )
        {
            c= HalftoneSmooth_C( CyanValue(cmykValue));
            m= HalftoneSmooth_M( MagentaValue(cmykValue));
            y= HalftoneSmooth_Y( YellowValue(cmykValue));
            k= HalftoneSmooth_K( BlackValue(cmykValue));
        }
```

```
else
    {
        c= HalftoneLineArt_C( CyanValue(cmykValue));
        m= HalftoneLineArt_M( MagentaValue(cmykValue));
        y= HalftoneLineArt_Y( YellowValue(cmykValue));
        k= HalftoneLineArt_K( BlackValue(cmykValue));
    }
}
```

[0041] Another aspect of the invention is directed to a method of providing printer status information. With reference to FIG. 4, a computer 400 is in communication with a printing device 500. The printing device 500 can be, for example, a printer or a print server. The computer 400 includes a browser 410, a hypertext transfer protocol (HTTP) server 420, a storage device 430, a spooler 440 and a driver interface 450. The printing device 500 includes a standard device driver 510, a simple network management protocol (SNMP) server 520 and a page description language (PDL) interpreter 530.

[0042] The browser 410 is in communication with HTTP server 420. The browser can be a standard web browser. The HTTP server 420 is in communication with the storage device 430 and the standard drive 450. The HTTP server 420 can be implemented as software. The storage device can be a disk drive. The spooler 440 is in communication with standard driver 450 and manages print jobs. The standard driver 450 can be a universal serial bus (USB) driver. The stander driver 450 of the computer 400 is in communication with printer 500 via the standard device driver 510. The standard device driver 510 is complementary to the device driver 450.

The standard device driver 510 is in communication with the SNMP server 520 and the PDL interpreter 530.

[0043] In operation, a user of the computer 400 requests the status of the printer 500 via the browser 410. The browser 410 sends an HTTP request to the HTTP server 420. In response the HTTP server 420 receives an HTML based status page stored in the storage device 430. Storing the HTML status pages in the storage device 430 reduces the traffic volume between the printing device 500 and the computer 400. The status page includes at least one field configured to receive information. The HTTP server 420 generates a SNMP request to retrieve the status information. The HTTP server 420 communicates the SNMP request to the standard driver 450, which in turn, forwards the request to the printing device 400. In one embodiment, the standard driver 450 uses a vendor specific command to send the SNMP request to the printing device 400.

[0044] The printing device 400 receives the SNMP request via the standard device driver 510. The SNMP server 520 receives the SNMP request from the standard device driver 510. In response the SNMP server 520 generates a response to the SNMP request. The response includes the status information requested by the user. The response is forwarded to the computer 400 via the standard device driver 510.

[0045] The computer 400 receives the response to the SNMP request at the standard driver 450, which, in turn, forwards the response to the HTTP server 420. The HTTP server 450 translates the SNMP response into a format that is useable by the printer status page, e.g., HTML. The translated response is inserted in the printer status page at the location defined by the field.

[0046] Having shown the preferred embodiments, one skilled in the art will realize that many variations are possible within the scope and spirit of the claimed invention. It is therefore the intention to limit the invention only by the scope of the claims.